Pentium® III Processor Specification Update

Release Date: June 1999

Order Number: 244453-004

The Pentium® III processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
March 1999	-001	This is the first Specification Update for Pentium® III processors.
April 1999	-002	Added Erratum E42. Deleted Erratum E16 and renumbered existing items. Corrected Errata table “Plans” column for E39. Updated the Pentium III Processor Identification Information table.
May 1999	-003	Updated the Pentium III Processor Identification Information table. Updated the Errata table by marking Errata E34, E35, and E40 as Fixed.
June 1999	-004	Updated the Pentium III Processor Identification and Package Information table. Added Erratum 43. Added Documentation Change E1. Added Specification Clarifications E1 and E2. Added Specification Change E3.

PREFACE

This document is an update to the specifications contained the *Pentium® III Processor at 450 MHz and 500 MHz* datasheet and the *Intel Architecture Software Developer's Manual*, *Volumes 1, 2 and 3* (Order Numbers 243190, 243191, and 243192, respectively). It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains Specification Changes, S-Specs, Errata, Specification Clarifications, and Documentation Changes.

Nomenclature

S-Spec Number is a five digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc., as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Specification Changes are modifications to the current published specifications for the Pentium® III processor. These changes will be incorporated in the next release of the appropriate documentation(s).

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the appropriate documentation(s).

Documentation Changes include errors (including typographical), or omissions from the current published specifications. These changes will be incorporated in the next release of the appropriate documentation(s).

Errata are design defects or errors. Errata may cause the Pentium III processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Identification Information

The Pentium III processor can be identified by the following values:

Family[1]	450 and 500- MHz[2]
0110	0111

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.

The Pentium III processor's second level (L2) cache size can be determined by the following register contents:

512-Kbyte Unified L2 Cache[1]	43h

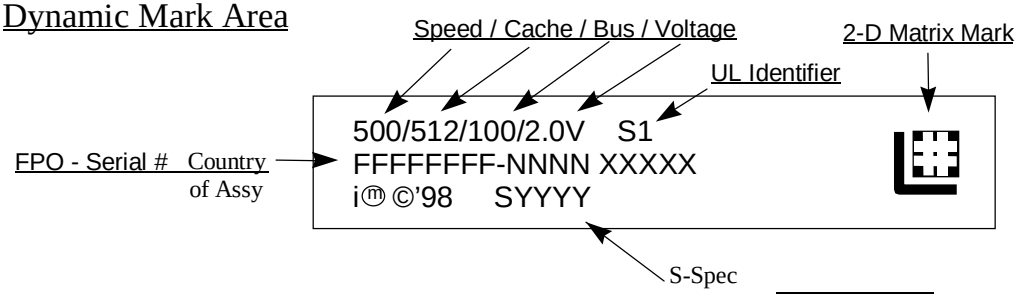
NOTE:

1. For the Pentium® III processor, the unified L2 cache size corresponds to a token in the EDX register after the CPUID instruction is executed with a 2 in the EAX register. Other Intel microprocessor models or families may move this information to other bit positions or otherwise reformat the result returned by this instruction; generic code should parse the resulting token stream according to the definition of the CPUID instruction.

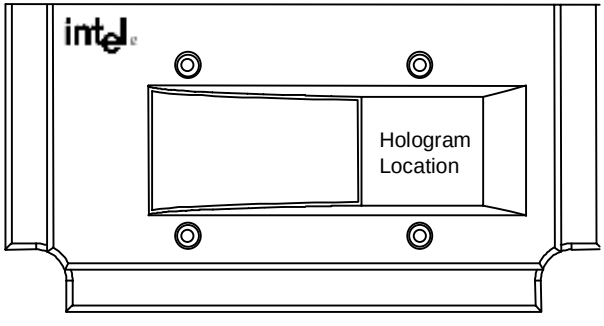
Specification Update for Pentium® III Processors

GENERAL INFORMATION

Pentium® III Processor and Boxed Pentium® III Processor 3 Line Markings



Pentium® III Processor Markings



Summary Table of Changes

The following table indicates the Specification Changes, Errata, Specification Clarifications, or Documentation Changes which apply to the Pentium III processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given processor stepping.
Doc:	Intel intends to update the appropriate documentation in a future revision.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
AP:	APIC related erratum.
SUB:	This column refers to errata on the Pentium® III processor substrate.

Shaded: This item is either new or modified from the previous version of the document.

Some of Intel's Specification Updates will be undergoing a numbering methodology change to reduce confusion when referring to errata which affect a specific product. Each Specification Update item will be prefixed with a capital letter to distinguish the product it refers to. The key below details the letters which will be used for the current Intel microprocessor Specification Updates:

A = Pentium® II processor

B = Mobile Pentium® II processor

C = Intel® Celeron™ processor

D = Pentium® II Xeon™ processor

E = Pentium® III processor

G = Pentium® III Xeon™ processor

H = Intel® Mobile Celeron™ processor

The Specification Updates for the Pentium® processor, Pentium® Pro processor, and other Intel products will not be implementing such a convention at this time.

Pentium® III Processor Identification and Package Information

S-spec	Core Stepping	CPUID	Speed (MHz) Core/Bus	L2 Size (Kbytes)	TagRAM/ Stepping	ECC/ Non-ECC	Processor Substrate Revision	Package and Revision	Notes
SL364	kB0	672	450/100	512	T6P-e/A0	ECC	D	S.E.C.C.2*	1, 2
SL365	kB0	672	500/100	512	T6P-e/A0	ECC	D	S.E.C.C.2*	1, 2
SL3CC	kB0	672	450/100	512	T6P-e/A0	ECC	D	S.E.C.C.2*	1, 2, 3
SL3CD	kB0	672	500/100	512	T6P-e/A0	ECC	D	S.E.C.C.2*	1, 2, 3
SL38E	kB0	672	450/100	512	T6P-e/A0	ECC	D	S.E.C.C	1, 2
SL38F	kB0	672	500/100	512	T6P-e/A0	ECC	D	S.E.C.C	1, 2
SL35D	kC0	673	450/100	512	T6P-e/A0	ECC	E	S.E.C.C.2*	1
SL37C	kC0	673	450/100	512	T6P-e/A0	ECC	E	S.E.C.C.2*	1, 3
SL35E	kC0	673	500/100	512	T6P-e/A0	ECC	E	S.E.C.C.2*	1
SL37D	kC0	673	500/100	512	T6P-e/A0	ECC	E	S.E.C.C.2*	1, 3
SL3F7	kC0	673	550/100	512	T6P-e/A0	ECC	E	S.E.C.C.2*	1

***** Unless otherwise noted, all Pentium® III processors in S.E.C.C.2 package have an OLGA package core.

NOTES:

1. These parts will only operate at the specified core to bus frequency ratio at which they were manufactured and tested. It is not necessary to configure the core frequency ratios by using the A20M#, IGNEE#, LINT[1]/NMI and LINT[0]/INTR pins during RESET.
2. These processors will not shut down automatically upon assertion of THERMTRIP#.
3. This is a boxed processor with an attached heatsink.

NO.	kB0	SUB	Plans	ERRATA
E1	X		NoFix	FP Data Operand Pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
E2	X		NoFix	Differences exist in debug exception reporting
E3	X		NoFix	FLUSH# servicing delayed while waiting for STARTUP_IPI in 2-way MP systems
E4	X		NoFix	Code fetch matching disabled debug register may cause debug exception
E5	X		NoFix	Double ECC error on read may result in BINIT#
E6	X		NoFix	FP inexact-result exception flag may not be set
E7	X		NoFix	BTM for SMI will contain incorrect FROM EIP
E8	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
E9	X		NoFix	Branch traps do not function if BTMs are also enabled
E10	X		NoFix	Checker BIST failure in FRC mode not signaled
E11	X		NoFix	BINIT# assertion causes FRCERR assertion in FRC mode
E12	X		NoFix	Machine check exception handler may not always execute successfully
E13	X		NoFix	MCE due to L2 parity error gives L1 MCACOD.LL
E14	X		NoFix	LBER may be corrupted after some events
E15	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
E16	X		NoFix	EFLAGS discrepancy on a page fault after a multiprocessor TLB shootdown
E17	X		NoFix	Near CALL to ESP creates unexpected EIP address
E18	X		NoFix	Memory type undefined for nonmemory operations
E19	X		NoFix	Infinite snoop stall during L2 initialization of MP systems
E20	X		NoFix	FP Data Operand Pointer may not be zero after power on or Reset
E21	X		NoFix	MOVD following zeroing instruction can cause incorrect result
E22	X		NoFix	Premature execution of a load operation prior to exception handler invocation
E23	X		NoFix	Read portion of RMW instruction may execute twice
E24	X		NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
E25	X		NoFix	Mixed cacheability of lock variables is problematic in MP systems
E26	X		NoFix	MOV with debug register causes debug exception
E27	X		NoFix	Upper four PAT entries not usable with Mode B or Mode C paging
E28	X		NoFix	Data breakpoint exception in a displacement relative near call may corrupt EIP
E29	X		NoFix	RDMSR and WRMSR to invalid MSR may not cause GP fault
E30	X		NoFix	SYSENTER/SYSEXIT instructions can implicitly load null segment selector to SS and CS registers
E31	X		NoFix	PRELOAD followed by EXTEST does not load boundary scan data
E32	X		NoFix	Far jump to new TSS with D-bit cleared may cause system hang

NO.	kB0	SUB	Plans	ERRATA
E33	X		NoFix	INT 1 instruction handler execution could generate a debug exception
E34	X		Fixed	COMISS/UCOMISS may not update EFLAGS under certain conditions
E35	X		Fixed	Transmission error on cache read
E36	X		NoFix	Potential loss of data coherency during MP data ownership transfer
E37	X		NoFix	Misaligned Locked access to APIC space results in hang
E38	X		NoFix	Floating point exception signal may be deferred
E39	X		NoFix	Memory ordering based synchronization may cause a livelock condition in mp systems
E40	X		Fixed	System bus address parity generator may report false AERR#
E41	X		NoFix	System bus ECC not functional with 2:1 ratio
E42	X		NoFix	Processor may assert DRDY# on a write with no data
E43	X		NoFix	GP# fault on WRMSR to ROB_CR_BKUPTMPDR6
E1AP	X		NoFix	APIC access to cacheable memory causes SHUTDOWN
E2AP	X		NoFix	2-way MP systems may hang due to catastrophic errors during BSP determination
E3AP	X		NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt

NO.	kB0	SUB	Plans	DOCUMENT CHANGES
E1	X		Doc	STPCLK# pin definition

NO.	kB0	SUB	Plans	SPECIFICATION CLARIFICATIONS
E1	X		Doc	MTRR initialization clarification
E2	X		Doc	Floating point opcode clarification

NO.	kB0	SUB	Plans	SPECIFICATION CHANGES
E1	X		Doc	FRCERR pin removed from specification
E2	X		Doc	Non-AGTL+ output leakage current change
E3	X		Doc	RESET# pin definition

ERRATA

E1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

PROBLEM: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

IMPLICATION: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

WORKAROUND: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E2. Differences Exist in Debug Exception Reporting

PROBLEM: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Pentium III processor, as described below:

CASE 1:

The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Pentium III processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

CASE 2:

In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4-Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

CASE 3:

If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

CASE 4:

If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

CASE 5:

When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

IMPLICATION: When debugging or when developing debuggers for a Pentium III processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

WORKAROUND: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E3. FLUSH# Servicing Delayed While Waiting for STARTUP_IPI in 2-way MP Systems

PROBLEM: In a 2-way MP system, if an application processor is waiting for a startup inter-processor interrupt (STARTUP_IPI), then it will not service a FLUSH# pin assertion until it has received the STARTUP_IPI.

IMPLICATION: After the 2-way MP initialization protocol, only one processor becomes the bootstrap processor (BSP). The other processor becomes a slave application processor (AP). After losing the BSP arbitration, the AP goes into a wait loop, waiting for a STARTUP_IPI.

The BSP can wake up the AP to perform some tasks with a STARTUP_IPI, and then put it back to sleep with an initialization inter-processor interrupt (INIT_IPI, which has the same effect as asserting INIT#), which returns it to a wait loop. The result is a possible loss of cache coherency if the off-line processor is intended to service a FLUSH# assertion at this point. The FLUSH# will be serviced as soon as the processor is awakened by a STARTUP_IPI, before any other instructions are executed. Intel has not encountered any operating systems that are affected by this erratum.

WORKAROUND: Operating system developers should take care to execute a WBINVD instruction before the AP is taken off-line using an INIT_IPI.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E4. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

PROBLEM: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However,

if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

IMPLICATION: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

WORKAROUND: The debug handler should clear breakpoint registers before they become disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E5. Double ECC Error on Read May Result in BINIT#

PROBLEM: For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Pentium III processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

IMPLICATION: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

WORKAROUND: Though the ability to drive BINIT# can be disabled in the Pentium III processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E6. FP Inexact-Result Exception Flag May Not Be Set

PROBLEM: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int

- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

IMPLICATION: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a "sticky" bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

WORKAROUND: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E7. BTM for SMI Will Contain Incorrect FROM EIP

PROBLEM: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

IMPLICATION: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E8. I/O Restart in SMM May Fail After Simultaneous MCE

PROBLEM: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Pentium III processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

IMPLICATION: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

WORKAROUND: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E9. Branch Traps Do Not Function If BTMs Are Also Enabled

PROBLEM: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

IMPLICATION: The branch traps and branch trace message debugging features cannot be used together.

WORKAROUND: If branch trap functionality is desired, BTMs must be disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E10. Checker BIST Failure in FRC Mode Not Signaled

PROBLEM: If a system is running in functional redundancy checking (FRC) mode, and the checker of the master-checker pair encounters a hard failure while running the built-in self test (BIST), the checker will tri-state all outputs without signaling an IERR#.

IMPLICATION: Assuming the master passes BIST successfully, it will continue execution unchecked, operating without functional redundancy. However, the necessary pull-up on the FRCERR pin will cause an FRCERR to be signaled. The operation of the master depends on the implementation of FRCERR.

WORKAROUND: For successful detection of BIST failure in the checker of an FRC pair, use the FRCERR signal, instead of IERR#.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E11. BINIT# Assertion Causes FRCERR Assertion in FRC Mode

PROBLEM: If a pair of Pentium III processors are running in functional redundancy checking (FRC) mode, and a catastrophic error condition causes BINIT# to be asserted, the checker in the master-checker pair will enter shutdown. The next bus transaction from the master will then result in the assertion of FRCERR.

IMPLICATION: Bus initialization via an assertion of BINIT# occurs as the result of a catastrophic error condition which precludes the continuing reliable execution of the system. Under normal circumstances, the master-checker pair would remain synchronized in the execution of the BINIT# handler. However, due to this erratum, an FRCERR will be signaled. System behavior then depends on the system specific error recovery mechanisms.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E12. Machine Check Exception Handler May Not Always Execute Successfully

PROBLEM: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

IMPLICATION: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

WORKAROUND: No workaround which would guarantee successful MCE handler execution under this condition has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E13. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

PROBLEM: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the Pentium III processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.

IMPLICATION: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E14. LBER May Be Corrupted After Some Events

PROBLEM: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the re-initialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

IMPLICATION: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E15. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

PROBLEM: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

IMPLICATION: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E16. EFLAGS Discrepancy on a Page Fault After a Multiprocessor TLB Shootdown

PROBLEM: This erratum may occur when the Pentium III processor executes one of the following read-modify-write arithmetic instructions and a page fault occurs during the store of the memory operand: ADD, AND, BTC, BTR, BTS, CMPXCHG, DEC, INC, NEG, NOT, OR, ROL/ROR, SAL/SAR/SHL/SHR, SHLD, SHRD, SUB, XOR, and XADD. In this case, the EFLAGS value pushed onto the stack of the page fault handler may reflect the status of the register after the instruction would have completed execution rather than before it. The following conditions are required for the store to generate a page fault and call the operating system page fault handler:

1. The store address entry must be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB before the store has completed. DTLB eviction requires at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation.
2. The page table entry for the store address must have its permissions tightened during the very small window of time between the DTLB eviction and execution of the store. Examples of page permission tightening include from Present to Not Present or from Read/Write to Read Only, etc.
3. Another processor, without corresponding synchronization and TLB flush, must cause the permission change.

IMPLICATION: This scenario may only occur on a multiprocessor platform running an operating system that performs “lazy” TLB shootdowns. The memory image of the EFLAGS register on the page fault handler’s stack prematurely contains the final arithmetic flag values although the instruction has not yet completed. Intel has not identified any operating systems that inspect the arithmetic portion of the EFLAGS register during a page fault nor observed this erratum in laboratory testing of software applications.

WORKAROUND: No workaround is needed upon normal restart of the instruction, since this erratum is transparent to the faulting code and results in correct instruction behavior. Operating systems may ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened or have a page fault handler that ignores any incorrect state.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E17. Near CALL to ESP Creates Unexpected EIP Address

PROBLEM: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

IMPLICATION: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

WORKAROUND: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an *indirect* call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E18. Memory Type Undefined for Nonmemory Operations

PROBLEM: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

IMPLICATION: Bus agents may decode a non-UC memory type for nonmemory bus transactions.

WORKAROUND: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

STATUS: For the steppings affected, see the Summary Table of Changes at the beginning of this section.

E19. Infinite Snoop Stall During L2 Initialization of MP Systems

PROBLEM: It is possible for snoop traffic generated on the system bus while a processor is executing its L2 cache initialization routine to cause the initializing processor to hang.

IMPLICATION: A DP (2-way) system which does not suppress snoop traffic while L2 caches are being initialized may hang during this initialization sequence.

WORKAROUND: System BIOS can create an execution environment which allows processors to initialize their L2 caches without the system generating any snoop traffic on the bus.

Below is a pseudo-code fragment, designed explicitly for a 2 processor system, that uses a serial algorithm to initialize each processor's L2 cache:

```
Suppress_all_I/O_traffic()
 K = 0;
 while (K <= 1)
 {
 /* Obtain current value of K. This forces both Temp and K into */
 /* the L1 cache. Note that Temp could also be maintained in a */
 /* general purpose register. */

 Temp = K;
 Wait_until_all_processors_are_signed_in_at_barrier()
 if ( logical_proc_APIC_id == K ) {
 {
 wait_10_usecs_delay_loop(); /* this time delay, required */
 /* in the worst case, allows */
 /* the barrier semaphore to */
 /* settle to shared state. */
 Initialize L2 cache
 K++
 }
 else
 while (Temp == K);
 }
 }
```

This algorithm prevents bus snoop traffic from the other processors, which would otherwise cause the initializing processor to hang. The algorithm assumes that the L1 cache is enabled (the Temp and K variables must be cached by each processor). Also, the Memory Type Range Register (MTRR) for the data segment must be set to WB (writeback) memory type.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E20. FP Data Operand Pointer May Not Be Zero After Power On or Reset

PROBLEM: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

IMPLICATION: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting in incorrect behavior of the software.

WORKAROUND: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E21. MOVD Following Zeroing Instruction Can Cause Incorrect Result

PROBLEM: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX™ technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
   or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
   or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
   or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
   or IMUL AX, BX, 1024 (opcode 69 /r iw)
   or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
   or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size may be affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX or IMUL instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

IMPLICATION: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

WORKAROUND: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD, IMUL-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX™ technology for operating on multiple variables. This would result in higher performance as well.

2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E22. Premature Execution of a Load Operation Prior to Exception Handler Invocation

PROBLEM: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation,
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending, or
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit set), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

If any of the above circumstances occur it is possible that the load portion of the instruction will have executed before the exception handler is entered.

IMPLICATION: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

WORKAROUND: Code which performs loads from memory that has side-effects can effectively workaround this behavior by using simple integer-based load instructions when accessing side-effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E23. Read Portion of RMW Instruction May Execute Twice

PROBLEM: When the Pentium III processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

IMPLICATION: This erratum has no effect if the memory targeted for the RMW instruction has no side-effects. If, however, the load targets a memory region that has side-effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

WORKAROUND: Hardware and software developers who write device drivers for custom hardware that may have a side-effect style of design should use simple loads and simple stores to transfer data to and from the device. Then, the memory location will simply be read twice with no additional implications.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E24. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

PROBLEM: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field, and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

IMPLICATION: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

WORKAROUND: When decoding the MC2_STATUS MSR, reverse the two error fields.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E25. Mixed Cacheability of Lock Variables Is Problematic in MP Systems

PROBLEM: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- If the processor which has that address cached has it cached in its L2 only.
- Other processors, meanwhile, issue back to back accesses to that same address on the bus.

IMPLICATION: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.

WORKAROUND: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a nonhomogenous memory model.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E26. MOV With Debug Register Causes Debug Exception

PROBLEM: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 14.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

IMPLICATION: With debug-register protection enabled (i.e., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

WORKAROUND: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E27. Upper Four PAT Entries Not Usable With Mode B or Mode C Paging

PROBLEM: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the Pentium III processor. However, in Mode B or Mode C paging, the upper four entries do not function correctly for 4-Kbyte pages. Specifically, bit seven of page table entries that translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE) are enabled, the processor forces this bit to zero when determining the memory type regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

IMPLICATION: Only the lower four PAT entries are useful for 4 KB translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may be used for large pages in Mode B or C paging.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E28. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

PROBLEM: If a data breakpoint is programmed at the memory location where the stack push of a near call is performed, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

IMPLICATION: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

WORKAROUND: Do not program a data breakpoint exception on the stack where the push for the near call is performed.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E29. RDMSR or WRMSR to Invalid MSR Address May Not Cause GP Fault

PROBLEM: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

IMPLICATION: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

WORKAROUND: Do not use invalid MSR addresses with RDMSR or WRMSR.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E30. SYSENTER/SYSEXIT Instructions Can Implicitly Load “Null Segment Selector” to SS and CS Registers

PROBLEM: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

IMPLICATION: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor’s behavior may become unpredictable, possibly resulting in system failure.

WORKAROUND: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E31. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

PROBLEM: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

IMPLICATION: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E32. Far Jump to New TSS With D-bit Cleared May Cause System Hang

PROBLEM: A task switch may be performed by executing a far jump through a task gate or to a new Task State Segment (TSS) directly. Normally, when such a jump to a new TSS occurs, the D-bit (which indicates that the page referenced by a Page Table Entry (PTE) has been modified) for the PTE which maps the location of the previous TSS will already be set, and the processor will operate as expected. However, if the D-bit is clear at the time of the jump to the new TSS, the processor will hang.

IMPLICATION: If an OS is used which can clear the D-bit for system pages, and which jumps to a new TSS on a task switch, then a condition may occur which results in a system hang. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused testing environment.

WORKAROUND: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E33. INT 1 Instruction Handler Execution Could Generate a Debug Exception

PROBLEM: If the processor's general detect enable flag is set and an explicit call is made to the interrupt procedure via the INT 1 instruction, the general detect enable flag should be cleared prior to entering the handler. As a result of this erratum, the flag is not cleared prior to entering the handler. If an access is made to the debug registers while inside of the handler, the state of the general detect enable flag will cause a second debug exception to be taken. The second debug exception clears the general detect enable flag and returns control to the handler which is now able to access the debug registers.

IMPLICATION: This erratum will generate an unexpected debug exception upon accessing the debug registers while inside of the INT 1 handler.

WORKAROUND: Ignore the second debug exception that is taken as a result of this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E34. COMISS/UCOMISS May Not Update Eflags Under Certain Conditions

PROBLEM: COMISS/UCOMISS instructions compare the least significant pairs of packed single-precision floating-point numbers and set the ZF, PF and CF bits in the EFLAGS register accordingly (the OF, SF and AF bits are cleared). Under certain conditions when a memory location is loaded into cache, the EFLAGS may not get set.

IMPLICATION: The result of the incorrect status of the EFLAGS may range from no effect to an unexpected application/OS behavior.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E35. Transmission Error on Cache Read

PROBLEM: During reads of the L2 cache, the processor may use certain L2 cache optimizations that may result in a data transmission error.

IMPLICATION: Data corruption caused by this erratum will result in unpredictable system behavior.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E36. Potential Loss of Data Coherency During MP Data Ownership Transfer

PROBLEM: In MP systems, processors may be sharing data in different cache lines, referenced as line A and line B in the discussion below. When this erratum occurs (with the following example given for a 2-way MP system with processors noted as ‘P0’ and ‘P1’), P0 contains a shared copy of line B in its L1. P1 has a shared copy of Line A. Each processor must manage the necessary invalidation and snoop cycles before that processor can modify and source the results of any internal writes to the other processor.

There exists a narrow timing window when, if P1 requests a copy of line B it may be supplied by P0 in an Exclusive state which allows P1 to modify the contents of the line with no further external invalidation cycles. In this narrow window P0 may also retire instructions that use the original data present before P1 performed the modification.

IMPLICATION: Multiprocessor or threaded application synchronization, required for low level data sharing, that is implemented via operating system provided synchronization constructs are not affected by this erratum. Applications that rely upon the usage of locked semaphores rather than memory ordering are also unaffected. This erratum does not affect uniprocessor systems. The existence of this erratum was discovered during ongoing design reviews but it has not as yet been reproduced in a lab environment. Intel has not identified, to date, any commercially available application or operating system software which is affected by this erratum. If the erratum does occur one processor may execute software with the stale data that was present from the previous shared state rather than the data written more recently by another processor.

WORKAROUND: Deterministic barriers beyond which program variables will not be modified can be achieved via the usage of locked semaphore operations. These should effectively prevent the occurrence of this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E37. Misaligned Locked Access to APIC Space Results in Hang

PROBLEM: When the processor's APIC space is accessed with a misaligned locked access a machine check exception is expected. However, the processor's machine check architecture is unable to handle the misaligned locked access.

IMPLICATION: If this erratum occurs the processor will hang. Typical usage models for the APIC address space do not use locked accesses. This erratum will not affect systems using such a model.

WORKAROUND: Ensure that all accesses to APIC space are aligned.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E38. Floating Point Exception Signal May Be Deferred

PROBLEM: A one clock window exists where a pending x87 FP exception that should be signaled on the execution of a CVTPS2PI, CVTPI2PS, or CVTTPS2PI instruction may be deferred to the next waiting floating point instruction or instruction that would change MMX register state.

IMPLICATION: If this erratum occurs the floating point exception will not be handled as expected.

WORKAROUND: Applications that follow Intel programming guidelines (empty all x87 registers before executing MMX technology instructions) will not be affected by this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section

E39. Memory Ordering Based Synchronization May Cause a Livelock Condition in MP Systems

PROBLEM: In an MP environment, the following sequence of code (or similar code) in two processors (P0 and P1) may cause them to each enter an infinite loop (livelock condition):

	P0			P1		
		MOV [xyz], EAX	(1)	wait1:	MOV EBX, [abc]	(2)
		.			CMP EBX, val1	(3)
		.			JNE wait1	(4)
		.				
		MOV [abc], val1	(6)		MOV [abc], val2	(5)
wait0:	MOV EBX, [abc]		(7)			
		CMP EBX, val2	(8)			
		JNE wait0(9)				

NOTE

The EAX and EBX can be any general-purpose register. Addresses [abc] and [xyz] can be any location in memory and must be in the same bank of the L1 cache. Variables "val1" and "val2" can be any integer.

The algorithm above involves processors P0 and P1, each of which use loops to keep them synchronized with each other. P1 is looping until instruction (6) in P0 is globally observed. Likewise, P0 will loop until instruction (5) in P1 is globally observed.

The P6 architecture allows for instructions (1) and (7) in P0 to be dispatched to the L1 cache simultaneously. If the two instructions are accessing the same memory bank in the L1 cache, the load (7) will be given higher priority and will complete, blocking instruction (1).

Instructions (8) and (9) may then execute and retire, placing the instruction pointer back to instruction (7). This is due to the condition at the end of the "wait0" loop being false. The livelock scenario can occur if the timing of the wait0 loop execution is such that instruction (7) in P0 is ready for completion every time that instruction (1) tries to complete. Instruction (7) will again have higher priority, preventing the data ([xyz]) in instruction (1) from being written to the L1 cache. This causes instruction (6) in P0 to not complete and the sequence "wait0" to loop infinitely in P0.

A livelock condition also occurs in P1 because instruction (6) in P0 does not complete (blocked by instruction (1) not completing). The problem with this scenario is that P0 should eventually allow for instruction (1) to write its data to the L1 cache. If this occurs, the data in instruction (6) will be written to memory, allowing the conditions in both loops to be true.

IMPLICATION: Both processors will be stuck in an infinite loop, leading to a hang condition. Note that if P0 receives any interrupt, the loop timing will be disrupted such that the livelock will be broken. The system timer, a keystroke, or mouse movement can provide an interrupt that will break the livelock.

WORKAROUND: Use a LOCK instruction to force P0 to execute instruction (6) before instruction (7).

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E40. System Bus Address Parity Generator May Report False AERR#

PROBLEM: The processor's address parity error detection circuit may fail to meet its frequency timing specification under certain environmental conditions. At the high end of the temperature specification and/or the low end of the voltage range, the processor may report false address parity errors.

IMPLICATION: If the system has AERR# drive enabled (bit [3] of the EBL_CR_POWERON resister set to '1') spurious address detection and reporting may occur. In some system configurations, BINIT# may be asserted on the system bus. This may cause some systems to generate a machine check exception and in others may cause a reboot.

WORKAROUND: Disable AERR# drive from the processor. AERR# drive may be disabled by clearing bit [3] in the EBL_CR_POWERON register. In addition, if the chipset allows, AERR# drive should be enabled from the chipset and AERR# observation enabled on the processor. AERR# observation on the processor is enabled by asserting A8# on the active-to-inactive transition of RESET#.

STATUS: For the processor part numbers affected see the "Pentium® III Processor Identification and Packaging Information" table at the General Information section.

E41. System Bus ECC Not Functional With 2:1 Ratio

PROBLEM: If a processor is underclocked at a core frequency to system bus frequency ratio of 2:1 and system bus ECC is enabled, the system bus ECC detection and correction will negatively affect internal timing dependencies.

IMPLICATION: If system bus ECC is enabled, and the processor is underclocked at a 2:1 ratio, the system may behave unpredictably due to these timing dependencies.

WORKAROUND: All bus agents that support system bus ECC must disable it when a 2:1 ratio is used.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E42. Processor May Assert DRDY# on a Write with No Data

PROBLEM: When a MASKMOVQ instruction is misaligned across a chunk boundary in a way that one chunk has a mask of all 0's, the processor will initiate two partial write transactions with one having all byte enables deasserted. Under these conditions, the expected behavior of the processor would be to perform both write transactions, but to deassert DRDY# during the transaction which has no byte enables asserted. As a result of this erratum, DRDY# is asserted even though no data is being transferred.

IMPLICATION: The implications of this erratum depend on the bus agent's ability to handle this erroneous DRDY# assertion. If a bus agent cannot handle a DRDY# assertion in this situation, or attempts to use the invalid data on the bus during this transaction, unpredictable system behavior could result.

WORKAROUND: A system which can accept a DRDY# assertion during a write with no data will not be affected by this erratum. In addition, this erratum will not occur if the MASKMOVQ is aligned.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E43. GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6

PROBLEM: Writing a '1' to unimplemented bit(s) in the ROB_CR_BKUPTMPDR6 MSR (offset 1E0h) will result in a general protection fault (GP#).

IMPLICATION: The normal process used to write an MSR is to read the MSR using RDMSR, modify the bit(s) of interest, and then to write the MSR using WRMSR. Because of this erratum, this process may result in a GP# fault when used to modify the ROB_CR_BKUPTMPDR6 MSR.

WORKAROUND: When writing to ROB_CR_BKUPTMPDR6 all unimplemented bits must be '0.' Implemented bits may be set as '0' or '1' as desired.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E1AP. APIC Access to Cacheable Memory Causes SHUTDOWN

PROBLEM: APIC operations which access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter shutdown after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. The second MCE signal will cause the Pentium III processor to enter shutdown.

IMPLICATION: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.

WORKAROUND: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E2AP. 2-Way MP Systems May Hang Due to Catastrophic Errors During BSP Determination

PROBLEM: In 2-way MP systems, a catastrophic error during the bootstrap processor (BSP) determination process should cause the assertion of IERR#. If the catastrophic error is due to the APIC data bus being stuck at electrical zero, then the system hangs without asserting IERR#.

IMPLICATION: 2-way MP systems may hang during boot due to a catastrophic error. This erratum has not been observed to date in a typical commercial system, but was found during focused system testing using a grounded APIC data bus.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

E3AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

PROBLEM: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the Pentium III processor despite the attempt to mask it via the LVT.

IMPLICATION: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.

WORKAROUND: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to *the Pentium® III Processor at 450 MHz and 500 MHz* datasheet, The *P6 Family of Processors Hardware Developer's Manual,* and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*.

E1. STPCLK# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual* and the *Pentium® III Processor at 450 MHz and 500 MHz* datasheet both have incorrect definitions of the STPCLK# pin in their alphabetical signal listing. These documents incorrectly state:

"The processor continues to snoop bus transactions and *service interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units and resumes execution."

They should state:

"The processor continues to snoop bus transactions and *may latch interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units, resumes execution, *and services any pending interrupts*."

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the *Pentium® III Processor at 450 MHz and 500 MHz* datasheet and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*.

E1. MTRR Initialization Clarification

The following sentence should be added to the end of the first paragraph of Section 9.11.5 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*: "The MTRRs must be disabled prior to initialization or modification."

E2. Floating Point Opcode Clarification

Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference,* provides detailed descriptions of each Intel Architecture instruction. For some instructions, the clarification phrase below needs to be either added to their existing "Comments" section or a "Comments" section needs to be created with the clarification phrase. The phrase is as follows:

> The (**Instruction** shown in the center column of the table below) instruction is actually a combination of two instructions – the wait instruction followed by (**Instruction** shown in the table). If the (**Instruction** shown in the table) instruction should fault in some way (e.g., page fault), the value of EIP that is passed to the fault handler will be equal to the EIP of the first instruction plus one (i.e., the EIP of the second of the pair of instructions). The FWAIT portion of the combined instruction will have completed execution and will typically not be, nor need to be, re-executed after the fault handler is completed.

The following table lists the affected instructions and the location of the clarification phrase:

Instruction Set Reference Section	Opcode	Instruction	Clarification	Page
FCLEX/FNCLEX-Clear Expectations	9B DB E2	FCLEX	Add "Comments" section with clarification phrase	3-177
FINIT/FNINIT-Initialize Floating-Point Unit	9B DB E3	FINIT	Add clarification phrase to existing "Comments" section	3-204
FSAVE/FNSAVE-Store FPU State	9B DD /6	FSAVE m94/108byte	Add clarification phrase to existing "Comments" section	3-237
FSTCW/FNSTCW-Store Control Word	9B D9 /7	FSTCW m2byte	Add "Comments" section with clarification phrase	3-250
FSTENV/FNSTENV-Store FPU Environment	9B D9 /6	FSTENV m14/28byte	Add "Comments" section with clarification phrase	3-253
FSTSW/FNSTSW-Store Status Word	9B DD /7	FSTSW m2byte	Add "Comments" section with clarification phrase	3-256
	9B DF E0	FSTSW AX		

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the *Pentium® III Processor at 450 MHz and 500 MHz* datasheet, and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*. All Specification Changes will be incorporated into a future version of the appropriate Pentium III processor documentation.

E1. FRCERR Pin Removed from Specification

The Pentium III processor will not use the FRCERR pin. All references to these pins will be removed from the specification. These references currently appear in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Appendix B.

E2. Non-AGTL+ Output Leakage Current Change

The CMOS, TAP, Clock and APIC Signal Group Output Leakage Current specification (I_{LO} in Table 6 of the *Pentium® III Processor at 450 MHz and 500 MHz* datasheet) should be changed from ±15 µA to ±30 µA.

E3. RESET# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual* and the *Pentium® III Processor at 450 MHz and 500 MHz* datasheet both have incorrect definitions of the RESET# pin in their alphabetical signal listing. These documents incorrectly state:

"RESET# must remain active for one microsecond for a 'warm' Reset; for a Power-on Reset, RESET# must stay active for at least one millisecond after $V_{CC_{CORE}}$ and CLK have reached their proper specifications."

They should state:

"For a Power-on or 'warm' reset, RESET# must stay active for at least one millisecond after $V_{CC_{CORE}}$ and CLK have reached their proper specifications."